



UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Corso di FONDAMENTI DI PROGRAMMAZIONE

Corso di Laurea in Ingegneria Informatica

Introduzione al Corso

Prof. Marco Mamei

Marco Mamei

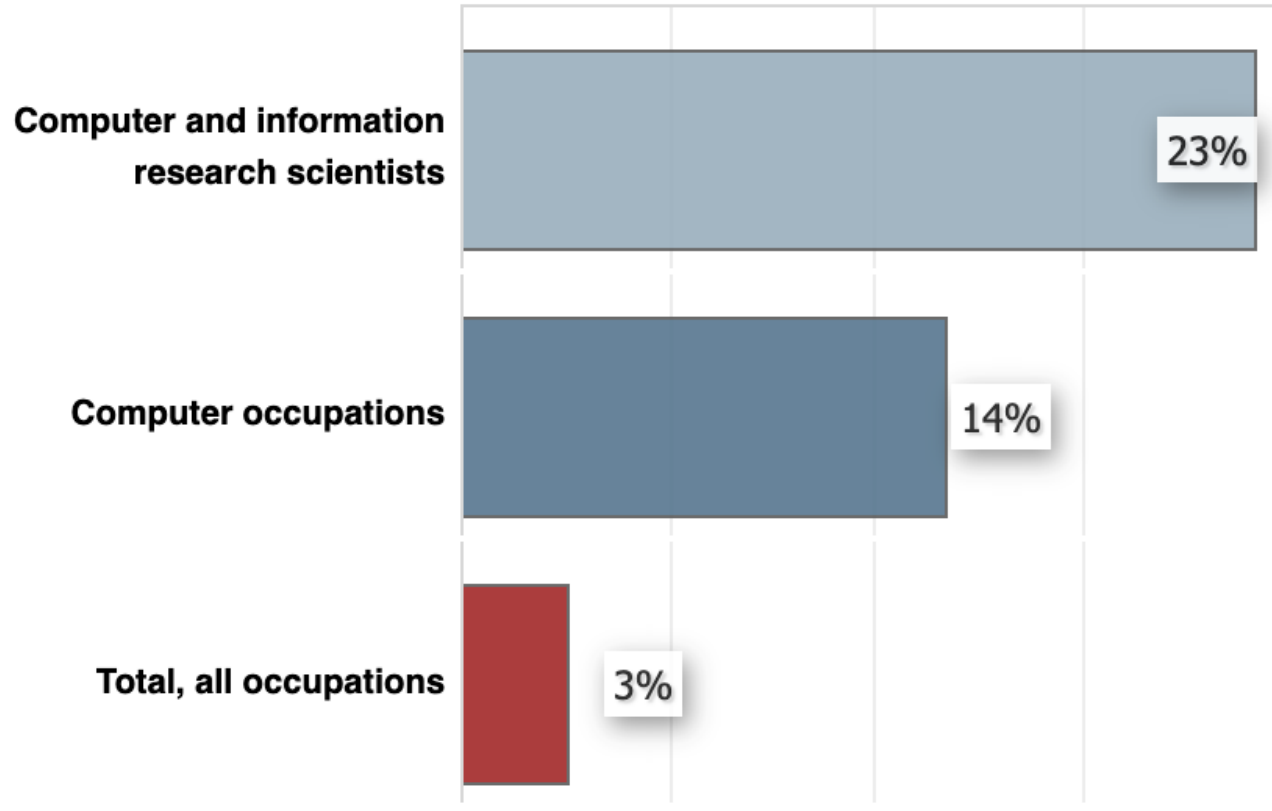
- Didattica
 - Fondamenti di Programmazione
 - Data Science and Management
 - Pervasive Computing e Servizi Cloud
- Ricerca
 - Analisi Dati Telecom e Internet of Things



Mercato del Lavoro

Computer and Information Research Scientists

Percent change in employment, projected 2022-32

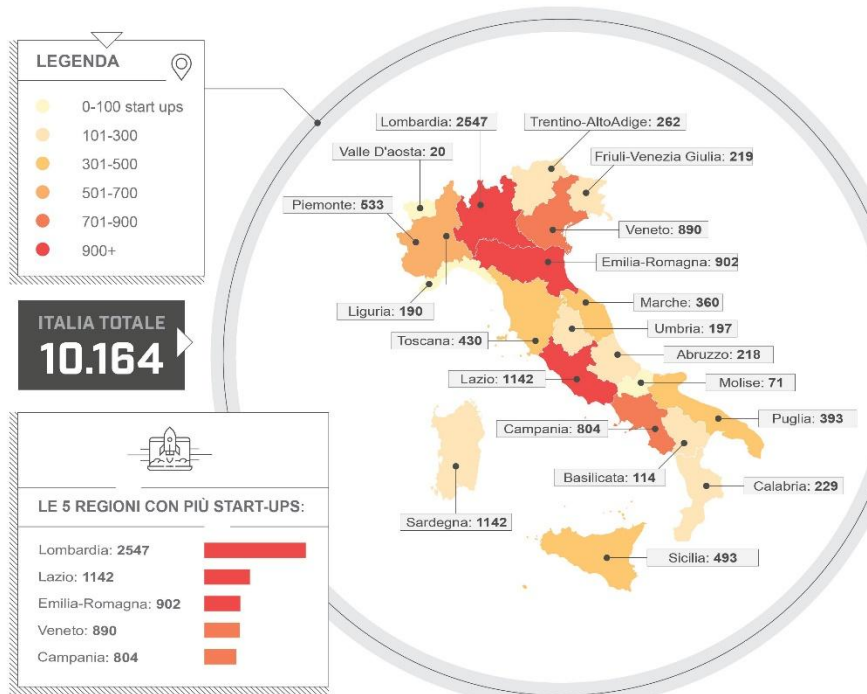


Note: All Occupations includes all occupations in the U.S. Economy.

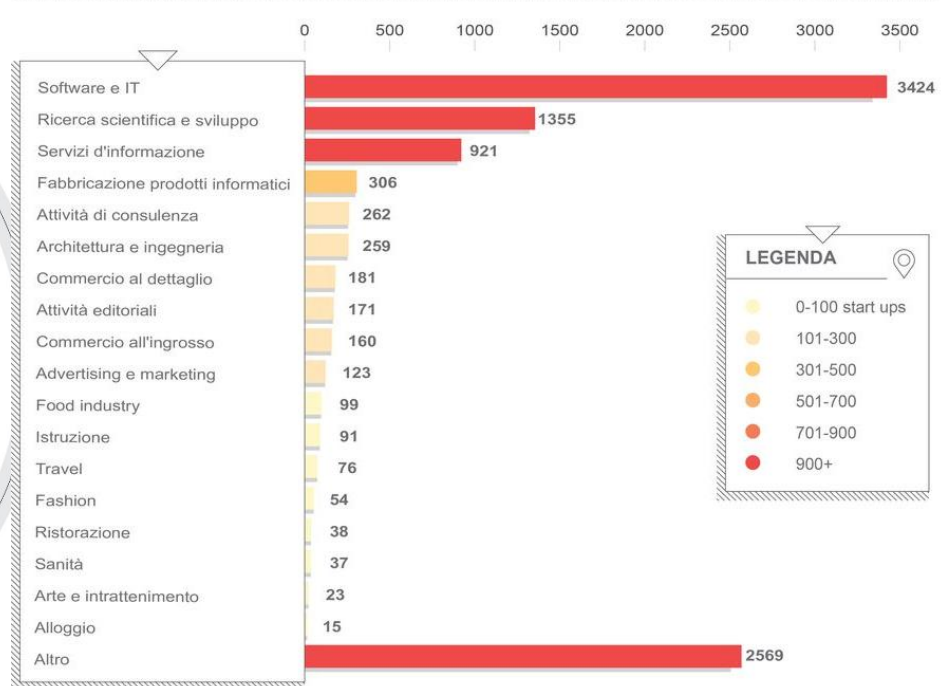
Source: U.S. Bureau of Labor Statistics, Employment Projections program

Creare Lavoro

MAPPA DELLE START-UP IN ITALIA PER REGIONE

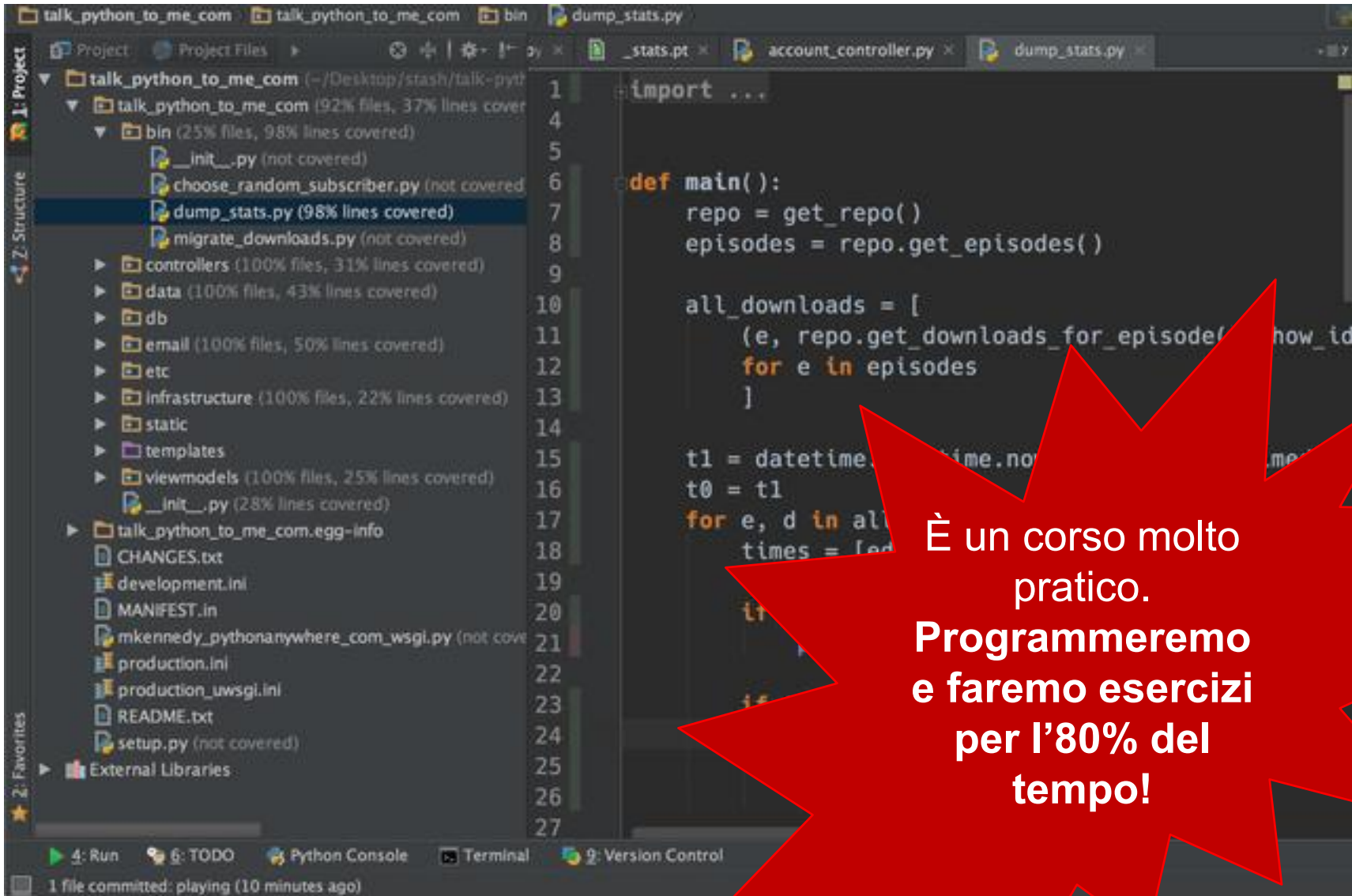


SETTORI DELLE START-UP



Startup in Italia (per segmento di attività)

Esempio di Programma



The image shows a screenshot of an IDE (likely PyCharm) with a project named 'talk_python_to_me_com'. The left sidebar displays the project structure, showing folders like 'bin', 'controllers', 'data', 'db', 'email', 'etc', 'infrastructure', 'static', 'templates', 'viewmodels', and 'talk_python_to_me_com.egg-info'. The 'bin' folder is expanded, showing files like 'dump_stats.py' (98% lines covered). The main editor window displays the code for 'dump_stats.py', which includes an 'import' statement and a 'def main():' function. The code is as follows:

```
1 import ...
2
3
4
5
6 def main():
7     repo = get_repo()
8     episodes = repo.get_episodes()
9
10    all_downloads = [
11        (e, repo.get_downloads_for_episode(e, how_id=...))
12        for e in episodes
13    ]
14
15    t1 = datetime.datetime.now()
16    t0 = t1
17    for e, d in all_downloads:
18        times = [e.timestamp() - t0 for t0 in d]
19
20    if ...
21
22
23
24
25
26
27
```

Overlaid on the right side of the IDE is a large red starburst graphic containing the following text:

**È un corso molto
pratico.
Programmeremo
e faremo esercizi
per l'80% del
tempo!**

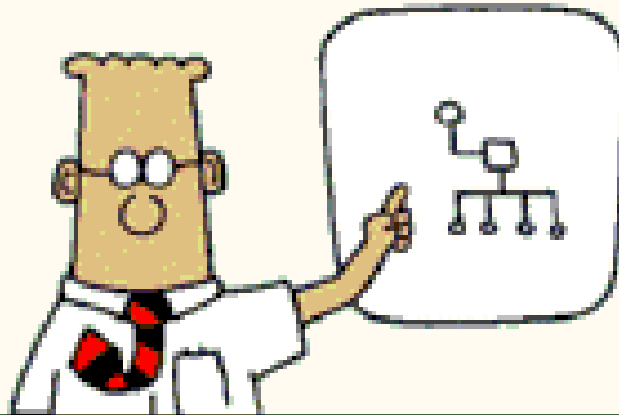
At the bottom of the IDE, there is a status bar showing '1 file committed: playing (10 minutes ago)' and a toolbar with icons for Run, TODO, Python Console, Terminal, and Version Control.

Problematiche del Corso (anni a.C)



Problematiche del Corso (anni a.C)

Ci sono domande?



www.dilbert.com scottadams@aol.com



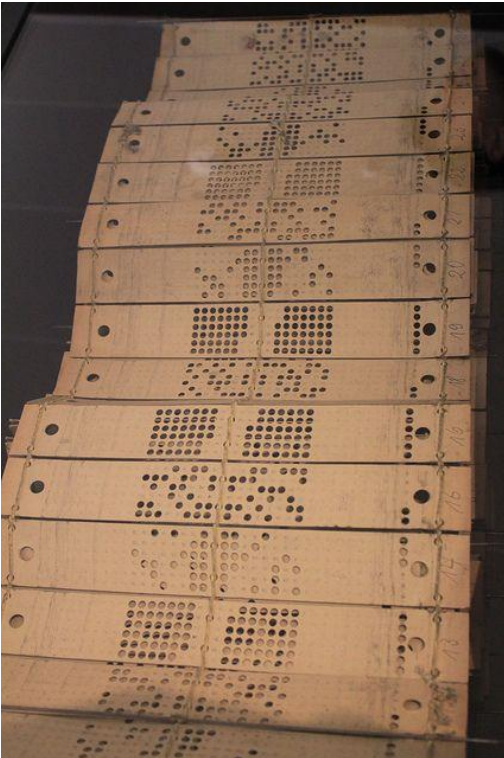
Nuove Problematiche (e Opportunità) del Corso (anni d.C)



btTV

**NVIDIA CEO JENSEN HUANG
SAYS....NO NEED TO LEARN CODING.**

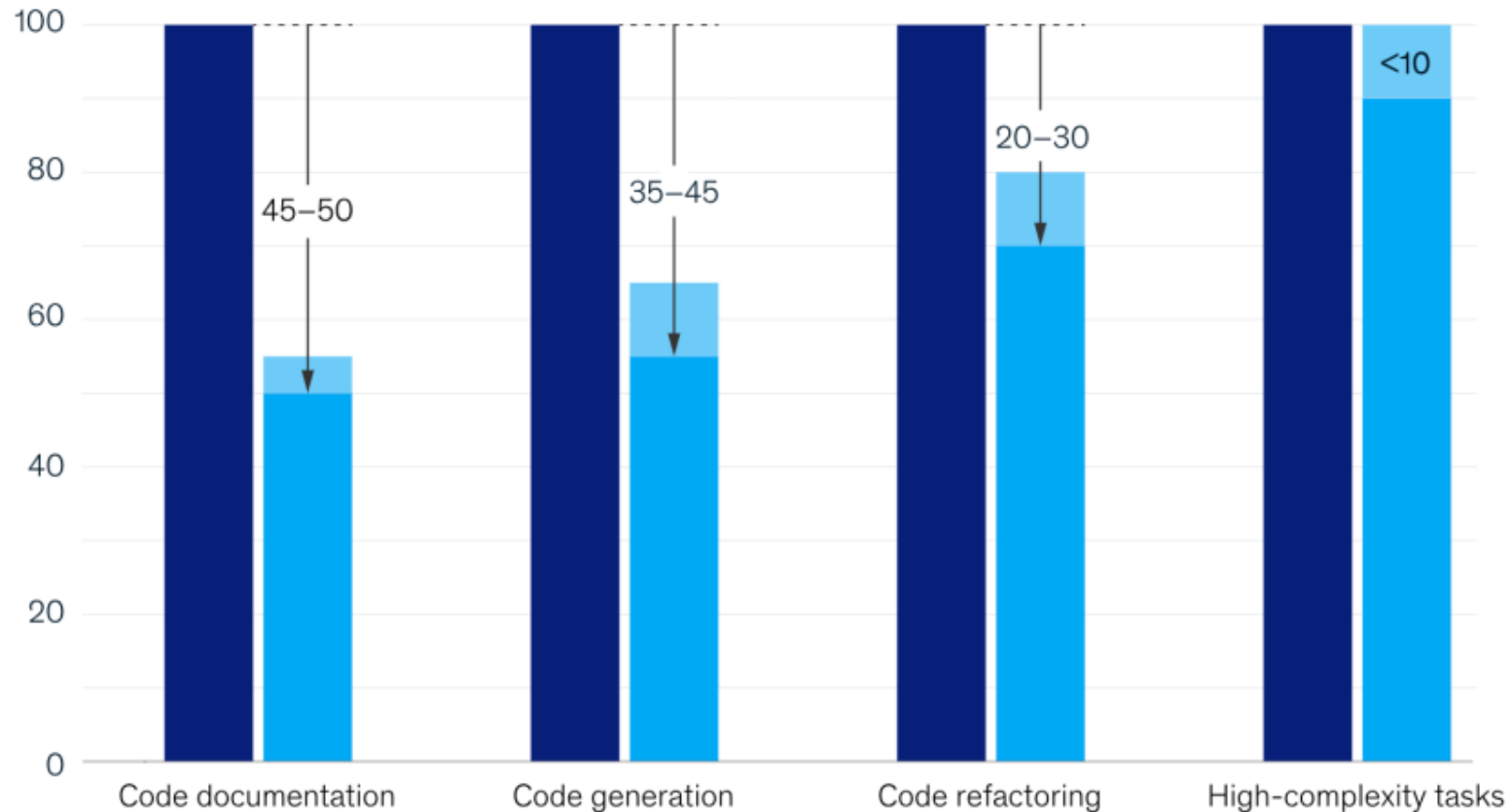
From 01 to High-level code



Generative AI can increase developer speed, but less so for complex tasks.

Task completion time using generative AI, %

Without generative AI With generative AI



Futuro e Presente della Programmazione

- Automatizzare compiti ripetitivi, come scrivere test o ottimizzare il codice.
- Ridurre la necessità di conoscenze tecniche dettagliate, spostando il focus verso la **comprensione del problema da risolvere**.
- Con **l'aumento della complessità dei sistemi**, i programmatori potrebbero dover specializzarsi in settori più specifici. Allo stesso tempo, sarà sempre più importante combinare competenze tecniche con **conoscenze di dominio**
- I programmatori diventeranno più simili a **consulenti tecnici, aiutando a personalizzare o integrare soluzioni**.
- Potrebbero **concentrarsi su attività più avanzate**.

Informazioni sul corso

- Docenti:

- Marco Mamei: marco.mamei@unimore.it
- Francesco Faenza: francesco.faenza@unimore.it
- Ricevimento: su appuntamento

- Durata del corso: circa 72 ore

- 48 ore – fondamenti di programmazione (M. Mamei)
- 24 ore – programmazione ad oggetti (F. Faenza)

- Il corso **non richiede conoscenze pregresse**, tranne quelle base di utilizzo del computer: accenderlo ☺, lanciare un programma, trovare, editare e copiare dei file

Materiale Didattico

□ **Materiale didattico.**

- Slides
- Esercizi
- Lezioni video-registrate (registrate nell'anno 3 a.C. - avanti ChatGPT)

□ Tutto il materiale sarà caricato su **Github**

□ **Libri**

- Learn AI-Assisted Python Programming, L. Porter, D. Zingaro, casa editrice Manning.

• **Ambiente di sviluppo**

- Python 3 (<https://www.python.org/downloads/>)
- Visual Studio Code (<https://code.visualstudio.com/>)
- Copilot (<https://github.com/features/copilot>) – (<https://github.com/education>)

- Guardare e riguardare le lezioni, o studiare sui libri serve come base iniziale, ma la cosa che dovete fare è esercitarvi a programmare ed esaminare codice.
- Provare a rifare gli esercizi proposti in classe con piccole/grandi variazioni.

Esame



Primi appelli a Gennaio-Febbraio. Poi di nuovo a Giugno-Settembre

Contenuti del corso

- Fondamenti (48 ore)
 - Introduzione (~3 ore)
 - Introduzione a Python (~6 ore)
 - Leggere codice Python (~15 ore)
 - Testing e Debugging (~3 ore)
 - Problem Decomposition (~6 ore)
 - GIT e Ambienti virtuali (~6 ore)
 - Esercizi (~9 ore)

- Programmazione ad Oggetti (24 ore)
 - OO Design (~6 ore)
 - Classi, oggetti, costruttore, metodi (~9 ore)
 - Polimorfismo, ereditarietà (~9 ore)



UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Corso di FONDAMENTI DI PROGRAMMAZIONE

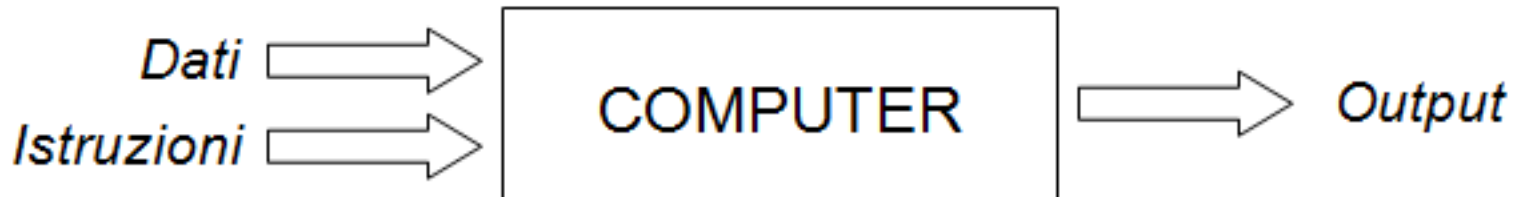
Corso di Laurea in Ingegneria Informatica

Introduzione alla Programmazione

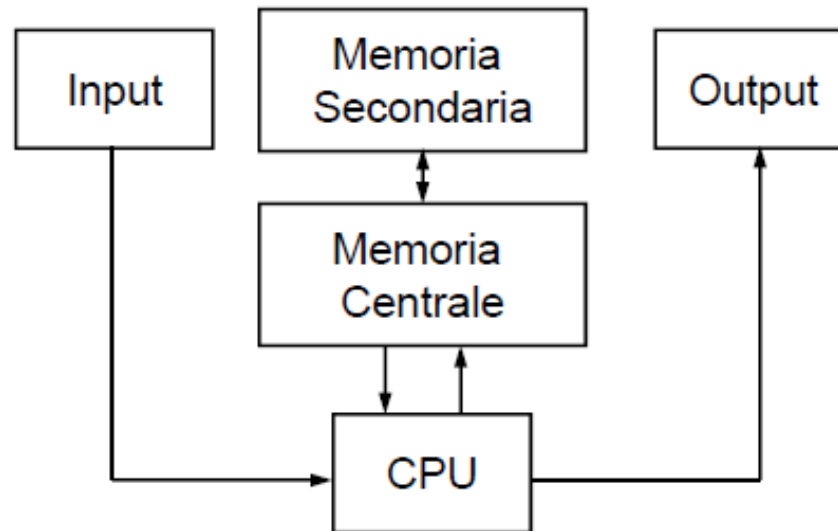
Prof. Marco Mamei

Computer o Calcolatore Elettronico

Computer, o calcolatore elettronico: è una macchina che produce dati in uscita (output) sulla base delle informazioni che riceve in ingresso (input) e di un insieme di istruzioni programma che specificano come operare sui dati



La macchina di Von Neumann

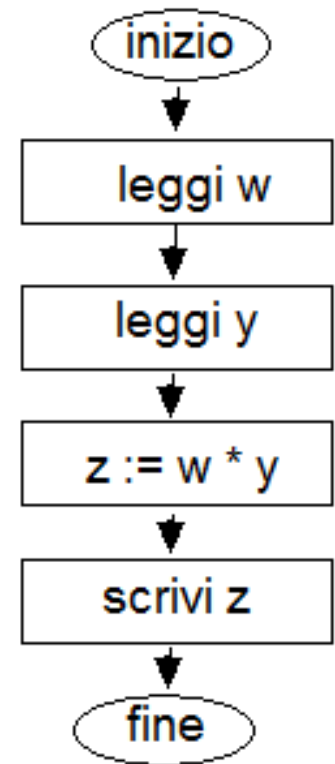


L'architettura di Von Neumann (1946) si compone di 3 parti fondamentali:

- **MEMORIA**: unità che mantiene le istruzioni che compongono il programma da eseguire e i dati che fungono da operandi per tali operazioni
- **CPU** (central processing unit) o processore: è composto da una unità di controllo per gestire le comunicazioni con tutte le unità del sistema e per gestire il prelievo delle istruzioni (fetch) e la loro esecuzione.
 - Nelle architetture moderne (più CPU - core, CPU specializzate – GPU)
- **INPUT/OUTPUT** le porte di interfaccia con l'esterno (e con i corrispondenti dispositivi periferici) .

Il Concetto di Algoritmo

- Il calcolatore elettronico per risolvere un problema utilizza un **algoritmo**, cioè un insieme di azioni (o **istruzioni**) che, eseguite secondo un ordine prestabilito, permettono di trovare il risultato cercato sulla base dei **dati in ingresso**
- Il concetto di algoritmo è uno dei concetti di base dell'intera matematica: i più semplici ed antichi algoritmi sono le regole per eseguire le operazioni dell'aritmetica elementare, formulate dal matematico arabo medioevale *Al-Khuwarizmi*, da cui deriva appunto il nome di algoritmo.
- Un computer è un rapidissimo esecutore di sequenze di istruzioni (gli algoritmi)



Esecuzione di un algoritmo sul computer

Algoritmo	<i>Procedura di trasformazione di un insieme di dati iniziali in un insieme di risultati finali mediante una sequenza di istruzioni non ambigue</i>
Linguaggio di programmazione	<i>Linguaggio (insieme di simboli e regole) per rappresentare le istruzioni di un algoritmo e la loro concatenazione</i>
Programma	<i>algoritmo scritto in un linguaggio di programmazione al fine di comunicare al calcolatore elettronico le azioni da eseguire (previa compilazione)</i>
Processo	<i>programma in esecuzione sul computer</i>

Algorithm

Your Python program:
`for i in range (10):
 print(i)`

The Python compiler
Converts your program into an intermediate form called bytecode

Python bytecode

The Python Virtual Machine
Converts the bytecode into executable machine code

1110 1010 0010 ... 0101
1110 0100 0111 ... 1011
1110 1000 0000 ... 0000
...
1110 0101 1110 ... 0001

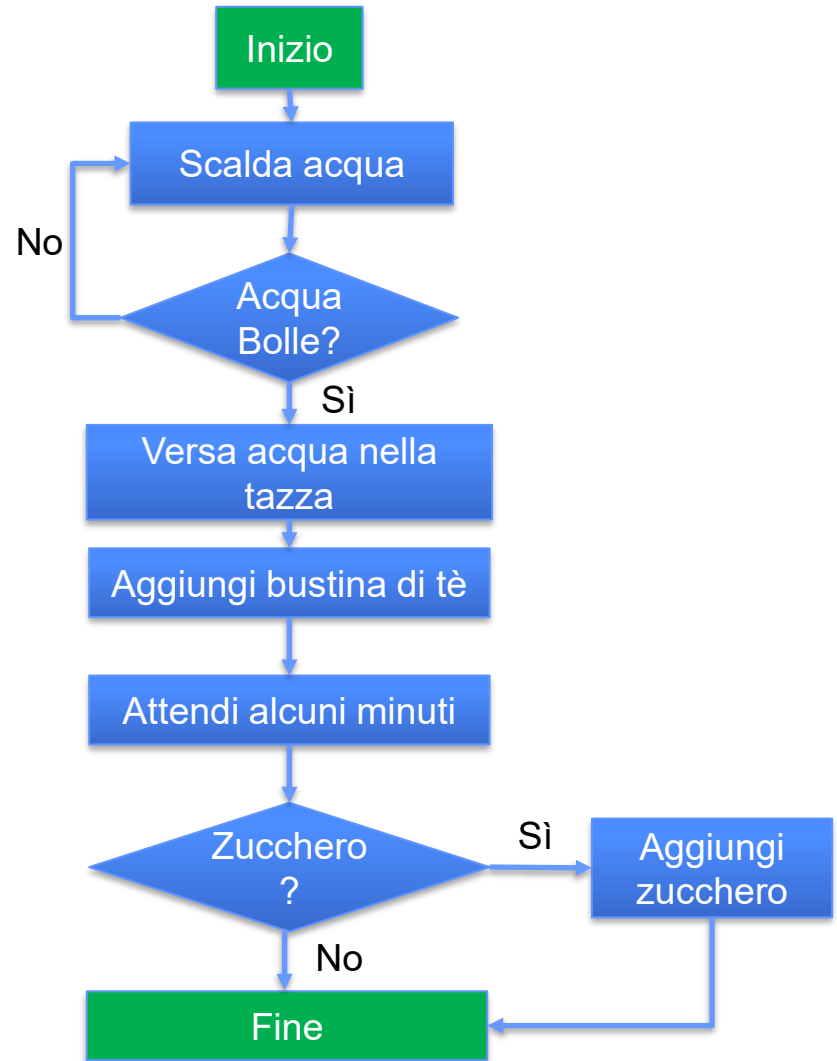
Your computer
Runs the machine code and displays the results of the program

Descrizione di un algoritmo: diagrammi di flusso

- I *diagrammi a blocchi* (*flow chart*) sono un formalismo grafico per rappresentare gli algoritmi
- Attraverso il diagramma a blocchi si può indicare l'ordine di esecuzione delle istruzioni
- Un particolare simbolo grafico detto **blocco elementare** è associato a ciascun tipo di istruzione
- I blocchi sono collegati fra loro tramite frecce che indicano il susseguirsi delle istruzioni

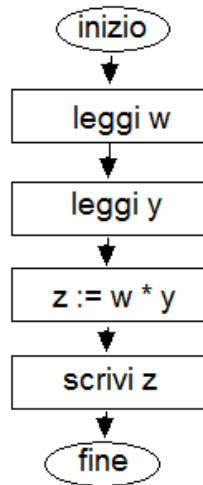
Esempio

1. Scaldare l'acqua
2. Se non è calda, continuare a scaldare (ciclo)
3. Versare nella tazza
4. Aggiungere bustina di tè
5. Attendere alcuni minuti
6. Se si vuole zucchero, aggiungerlo (condizione)



Fondamenti di Programmazione Tradizionale

Algorithm



Python Code

```
w = float(input("Inserisci W\n"))
y = float(input("Inserisci Y\n"))
z = w * y
print(z)
```

The Python compiler
Converts your program into an intermediate form called bytecode

Python bytecode

The Python Virtual Machine
Converts the bytecode into executable machine code

```
1110 1010 0010 ... 0101
1110 0100 0111 ... 1011
1110 1000 0000 ... 0000
...
1110 0101 1110 ... 0001
```

Your computer
Runs the machine code and displays the results of the program

«Nuovo» Fondamenti di Programmazione

You type a prompt like:

```
# Output the numbers from 0 to 9
```



Copilot sends your prompt to the large language model on the internet.



OpenAI's GPT large language model interprets your prompt and generates some code.



```
for i in range (10):  
    print(i)
```

Cosa Dobbiamo Imparare? (1/2)

- If Copilot can write our code, explain it, and fix bugs in it, are we just done? Do we just tell Copilot what to do and celebrate?
- No. First, Copilot can make mistakes. The code it gives us might be syntactically correct, but sometimes it doesn't do what we want it to do. We need to be vigilant to catch when Copilot makes these mistakes.
- Second, although some of the skills that programmers rely on (e.g., writing correct syntax) will decrease in importance, other skills remain critical. For example, you can't throw a huge task at Copilot like, "Make a video game. Oh, and make it fun." Copilot will fail. **Instead, we need to break down such a large problem into smaller tasks that Copilot can help us with. How do we break a problem down like that?** Not easily, it turns out. Humans need to develop this key skill when engaging in conversations with tools like Copilot.

Cosa Dobbiamo Imparare? (2/2)

- Other skills, believe it or not, may take on even more importance with Copilot.
- **Testing code** has always been a critical task in creating high-quality code. We know a lot about testing code written by humans because we know where to look for typical problems. What about code written by AI, where 20 lines of flawless code could hide 1 line so absurd that we likely wouldn't expect it there? We don't have experience with that. We need to test even more carefully than before
- We also need to know how to fix mistakes when the code is wrong. This process is called **debugging** and is still essential, particularly when Copilot gives you code that is close to correct but not quite there yet.
- **Understanding** code is important to work in other projects (**legacy**)
- Finally, some required skills are entirely new. The main one here is called **prompt engineering**, which involves how to tell Copilot what to do.